Chapter 35

PIC1650: Chip Architecture and Operation

Frank M. Gruppuso

I. Introduction and Design Goals

The PIC1650 is an MOS/LSI circuit array containing RAM, I/O, a central processing unit, and a customer-defined ROM on a single chip. General Instrument (GI) architectured the PIC (Programmable Intelligent Controller) in 1976 to satisfy the need for a low-level, easy-to-use microcontroller. The only other microcomputer available at the time was the calculator-based design TMS1000, and it was felt that a much more powerful machine could be built around a general-purpose–register, minicomputer-like architecture. Thus was laid the groundwork for the PIC1650.

The PIC is fabricated in an N-channel MOS–process technology that permits fabrication of both enhancement- and depletion-mode transistors. Depletion-mode transistors allow low-voltage (5-volt) operation and, when used as internal load resistors, offer much better speed-power performance than enhancement-mode transistors used in a similar fashion.

As a controller, the PIC chip was designed to emphasize bit, byte, and register-transfer operations. Its main objectives would be to perform logical processing, basic code converting, and formatting, and to generate fundamental timing and control signals for various subservient I/O devices [PIC 1979*a*,*b*]. The emphasis was placed on the ability to provide control and interface functions rather than computing functions. The PIC was seen as a key element to providing so-called intelligence to long-established non-computer, small-system designs which, as it turned out, were mostly electromechanical in nature. Some of the initial proposals were for applications in vending machines, small dot-matrix impact printers, and metered mailing systems.

The following are several key issues which motivated the architecture and logic design.

- Wide instruction word. It was felt that a 12-bit-wide instruction word that was wider than the 8-bit data word afforded both simplicity (and thus compactness) of chip design and ease of user-programmability. All instructions were therefore designed to be one word long; this kept the control logic simple since no multiple fetches had to be made from program memory to execute even the most complex instructions. Also, a 12-bit instruction word allows every register to be directly addressed by the program. It further permits literals in program ROM to be accessed at the same time as the instruction op code. For example, in a machine with an 8-bit-wide instruction word, a load immediate instruction would normally take two 8-bit ROM words fetched and executed in two instruction cycles. In the PIC1650, the equivalent instruction only occupies one 12-bit ROM word in memory and executes in one cycle.

- General-register architecture. Another aspect of the design that was considered important was the general-purpose nature of the register array: the program counter (PC), every I/O register, and most other specialized registers occupy an address in the register array address space. This permits every instruction that can operate on a general-purpose register to operate, say on an I/O file register or the PC. In the case of the program counter, for example, the instruction MOVW F2 (move the contents of the working register to the PC) is actually a computed GOTO instruction.

- Minimal parts count. It was envisioned that the PIC would be applied in areas that would be cost-sensitive from a systems viewpoint. Thus, efforts were taken to minimize the amount of external outboard circuitry. A single-pin oscillator whose frequency of operation was determinable by a single resistor and capacitor was designed. A second power supply, V_{XX} was added to drive the output buffers of the processor. It was not expected that TTL gates would be the only loads that the microcomputer I/O lines were ever going to see. Discrete switching transistors, coils, and large LED displays represent only a few of the different kinds of external circuits it was felt the PIC chip had to be capable of interfacing with. If V_{XX} is varied externally from 5 V to, say, 9 V, the output buffer transistors behave as voltage-controlled resistors. This allows any interface between the PIC chip and the outside world to be more effectively matched. Section VI of this chapter describes an application using this pin.

- Direct Bit Set/Clear/Test instructions. In view of the PIC's overall architectural goal of being a controller, it was highly desirable for the processor to be able to directly set, clear, and test individual bits in any register without forcing the user to program the usual "mask with a literal" coding sequences. Instead, the chip performs these functions internally. Thus, to execute the Bit Set instruction on bit 2 of a particular register, for example, the processor internally sets up the mask B'00000100' and performs a logical OR between this mask and the register.

- Wide operating-voltage range. Soon after the release of the PIC1650 to the marketplace, it became apparent that a number of applications were found which required battery operation (e.g., electronic hand-held games and digital scales) or, more generally, a wide operating-voltage range. A wide operating-voltage–range chip could tolerate a less critical and, hence, less costly external power supply. Thus, GI initiated a design effort that generated the "A" series of PIC chips—PIC1650A, PIC1655A—that are identical to the original except that the operating voltage range was increased from 4.75–5.25 V to 4.5–7 V. As four C cell batteries fully charged produce 6.8 V in series, 7 V was chosen as the upper limit.

Several versions of the PIC1650 have been architectured which, among other things, vary according to number of I/O lines, RAM size, and ROM size. These are enumerated in Table I.

Applications using the PIC series have centered around those where a single-chip microcomputer could perform systems functions at a lower cost than non-computer solutions presently available or, alternatively, provide extra features which heretofore would have been prohibitively expensive without a microcomputer. Present applications that use the PIC chip include:

- Digital-readout weight scale. In this application, weight is converted to a digital pulse train via a front-end transducer circuit. The pulses are applied to the RTCC (Real Time Clock Counter) input. The ROM program computes the difference between the frequency with the weight applied and the no-weight frequency (thereby providing for auto-zero correction) and converts the difference to a 4-digit BCD number which is subsequently displayed. To save multiplexing costs, all thirty-two I/O lines drive the display directly.
- Auto-dialer telephone system. This system is capable of storing and retrieving sixteen 10-digit telephone numbers. Here, the PIC chip processes command codes, which are entered through the keyboard, and drives outboard CMOS RAM, which stores the actual digits.
- Motor control. In this application, the PIC chip serves as a feedback element in a constant-speed motor control system. The microcomputer senses the present speed of the motor and adjusts the firing pulse to an external SCR, which, in turn, drives the motor. The high instruction rate permits precise control over a wide range of speeds. Typical applications of this system are found in industrial drill presses and hand drills.
- Consumer electronics. In the consumer arena, the PIC has been programmed into a variety of electronic games. The PIC is quite efficient in the area of sound generation: the high instruction rate permits higher-frequency sounds and thus more complex sounds than would be possible with a slower processor. Other areas of consumer electronics use the aforementioned motor control technique in household mixers, blenders, and food processors. The PIC has also been designed into appliances requiring time controlling, such as microwave oven timing.

Figure 1 shows a functional block diagram of the PIC1650. All data elements—arithmetic logic unit, register file array, I/O registers—are connected via an internal 8-bit bidirectional bus.

Table 1 Family of PIC Architectures

	PIC1650A	*PIC1655A*	*PIC1670*	*PIC1645*	*PIC1656*
Technology	NMOS	NMOS	NMOS	NMOS	NMOS
Number of pins per package	40	28	40	18	28
Cycle time (μsec)	4	4	4	4	4
Data path width (bits)	8	8	8	8	8
Instruction word width (bits)	12	12	12	12	12
Program ROM size (12-bit bytes)	512	512	1024	256	512
Number of fixed instructions	30	30	33	31	31
Data storage RAM size (8-bit bytes)	32	32	64	24	32
Operating-voltage supply range (volts)	4.5–7.0	4.5–7.0	4.5–7.0	4.5–7.0	4.5–7.0
Interrupt capability	No	No	Yes	Yes	Yes
Levels of pushdown stack	2	2	4	3	3
I/O configuration (registers)					
Input/output	4 8-bit	1 8-bit	4 8-bit	1 4-bit	1 8-bit
Input only		1 4-bit		1 4-bit	1 4-bit
Output only		1 8-bit		1 4-bit	1 8-bit
Date of introduction	1978	1978	Planned 1979	Planned 1979	1979

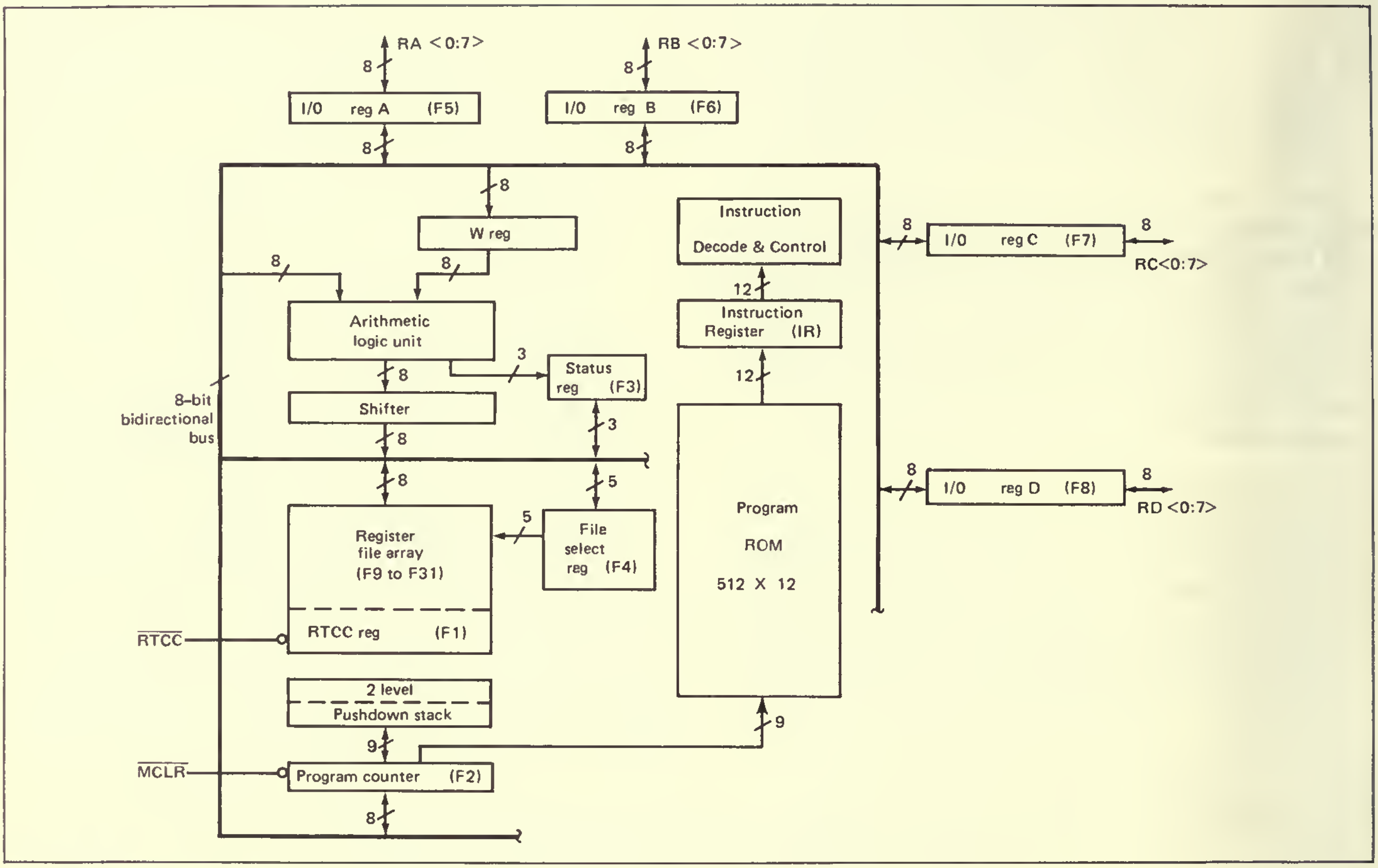


Fig. 1. PIC1650 block diagram.

Descriptions of these various elements appear in the following sections.

II. Mp State

The Program ROM contains 6144 bits organized as 512 twelve-bit words. RAM storage consists of 32 eight-bit registers, all of which are addressable by instructions contained in the program ROM. These registers are divided into two functional groups: operational registers and general-purpose registers. The general registers are addressed as F9 to F31 and contain data and control information. These registers are all located in a contiguous block labeled "Register File Array" in Fig. 1. The operational registers, F0 to F8, are scattered throughout the chip, and not only are they addressable by the program, but they also perform special functions described in Sec. III of this chapter.

III. Pc State

The register file arrangement is delineated as follows:

a F0. F0 is not a physically implemented register. Rather, it is used as an indirect register-select mechanism; when F0 is specified in the register file field of an instruction, the PIC will use the contents of F4 to select the register to be used in that instruction.

b F1<7:0>\Real.Time.Clock.Counter.Register. This register counts external events by incrementing on the falling edge of the RTCC pin. This register can also be loaded and read under program control.

c F2<8:0>\Program.Counter (PC). The program counter points to the next instruction to be executed in memory. This register is 9 bits wide to address the 512-word ROM, but only the low-order 8 bits can be written to or read from by the program. The ninth bit can be considered a page bit and can only be altered by a GOTO instruction. The PC is initialized to 777_8 upon a low-to-high transition of the MCLR input pin. It increments normally thereafter except as modified by the program via the use of the CALL, RETURN or other, similar instructions.

d F3<2:0>\Status.Word.Register. This register contains status bits which are modified as a result of arithmetic operations.

C<>\ Carry.bit:= F3 <0>. Stores the carry out of the most significant bit of the resultant of an arithmetic operation. This bit is also used as a link for rotate instructions.

DC<>\ Digit.Carry := F3<1>. Stores the carry out of the fourth low-order bit (bit 3) of the resultant of an arithmetic operation. The bit is useful in processing decimal data.

Z<>\Zero := F3 <2>. This bit is set if the resultant of an arithmetic operation is zero and cleared if the resultant is not zero.

Since these bits constitute a file register, they can also be modified under program control. However, to avoid a conflict between altering the status flags under program control and altering the status flags as a result of arithmetic operations, F3 can only be modified under program control by either the BIT SET or BIT CLEAR instruction.

e F4<7:0>\ File.Select.Register (FSR). Only the low-order 5 bits are used in this register. The FSR is used in generating effective file register addresses under program control. When this register is directly addressed as a file, all 5 bits can be written to and read from. The upper 3 bits read as a logic "1."

f F5<7:0>\ Input.Output.Register A.

g F6<7:0>\ Input.Output.Register.B.

h F7<7:0>\ Input.Output.Register.C.

i F8<7:0>\ Input.Output.Register.D.

j F9<7:0>–F31<7:0>. Twenty-three general-purpose registers.

k W<7:0>\ Working.Register. The accumulator.

l Stack [1:0]<8:0>. Two registers that store return addresses for use in CALL and RETURN instructions.

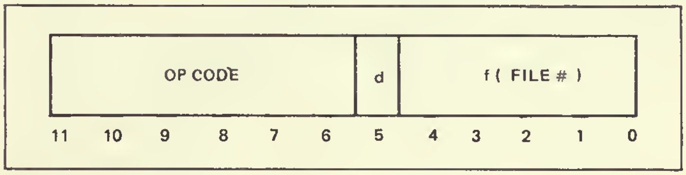
m IR<11:0>\ Instruction.Register. A 12-bit register that stores the instruction currently being executed by the PIC.

Note that neither register of the pushdown stack can be directly accessed by the program. When a CALL instruction is executed, the contents of the program counter (which is already pointing to the next instruction after the CALL) are pushed into the top register of the pushdown stack. The top register's former contents are pushed onto the second register in the stack. Any prior data in this second register is lost, thereby limiting the amount of subroutine nesting to two. The RETURN instruction (mnemonic RETLW) functions in reverse fashion: the top register of the stack replaces thc current PC while the second register of the stack replaces the stack top. The contents of the second register remain unchanged.

IV. Instruction Set

Table 2 summarizes the PIC1650 instruction set. Each instruction is a 12-bit word divided into an op code field which specifies the instruction type and one or more fields which select the operand data source and destination. The instruction set is broken into three different formats: general file register operations, bit-level file register operations, and literal and control operations.

Instruction Format I: General File Register Operations



This format has a 6-bit op code field, a 5-bit register select field, and a single-bit destination field. The 5-bit register select field can directly access any one of the 32 file registers (F0 through F31). Instructions in this format will specify either a single source operand—a file register—or two source operands—a file register and the W register.

Two-source operand instructions include SUBTRACT, Inclusive OR, AND, Exclusive OR, ADD, and MOVE. Single-source operand instructions include CLEAR, DECrement, COMplement, INCrement, Rotate Right, Rotate Left, and SWAP. For all instructions in this format, however, the destination bit (bit 5) will specify where the result of the operation will be placed. If the destination bit equals 1, the result will be placed in the file register originally specified as the source; if the destination bit equals 0, the result will be placed in the W register.

Two other instructions in this format permit compact coding in the case of software timing loops. *Decrement file, skip if zero* (DECFSZ) decrements the source file, and if the result of the decrement operation is zero, then the next instruction after

Table 2 PIC 1650 Instruction-Set Summary

In the following PIC instruction descriptions "k" represents an eight-bit constant or literal value, "f" represents a file register designator and "d" represents a destination designator. The file register designator specifies which one of the 32 PIC file registers is to be utilized by the instruction. The destination designator specifies where the result of the operation performed by the instruction is to be placed. If "d" is zero, the result is placed in the PIC W register. If "d" is one, the result is returned to the file register specified in the instruction. If the "d" operand is omitted, the f register is assumed as the destination. "f" and "d" may be numbers, characters, or symbols as described in the PIC Assembler and PIC Simulator instructions. A "b" field specifies the bit number within an 8-bit register, "C" represents the carry bit, "Z" represents the zero bit, and "DC" represents the digit carry bit.

General file register operations

(6–11)	(5)	(0–4)
OP CODE	d	f (FILE #)

for d = 0, f→W
d = 1, f→f

Instruction (octal)				*Name*	*Syntax*		*Operation*	*Status affected*
000000	0	00000	(0000)	No Operation	NOP	...	.....	None
000000	1	fffff	(0040)	Move W to f†	MOVWF	f	W→f	None
000001	0	fffff	(0100)	Clear W	CLRW	-	O→W	Z
000001	1	fffff	(0140)	Clear f	CLRF	f	O→f	Z
000010	d	fffff	(0200)	Subtract W from f	SUBWF	f, d	f−W→d	C, DC, Z
000011	d	fffff	(0300)	Decrement f	DECF	f, d	f−1→d	Z
000100	d	fffff	(0400)	Inclusive OR W and f	IORWF	f, d	W∨f→d	Z
000101	d	fffff	(0500)	AND W and f	ANDWF	f, d	W∧f→d	Z
000110	d	fffff	(0600)	Exclusive OR W and f	XORWF	f, d	W⊻f→d	Z
000111	d	fffff	(0700)	Add W and f	ADDWF	f, d	W+f→d	C, DC, Z
001000	d	fffff	(1000)	Move f	MOVF	f, d	f→d	Z
001001	d	fffff	(1100)	Complement f	COMF	f, d	$\bar{f}$→d	Z
001010	d	fffff	(1200)	Increment f	INCF	f, d	f+1→d	Z
001011	d	fffff	(1300)	Decrement f, Skip if Zero	DECFSZ	f, d	f−1→d, skip if Zero	None
001100	d	fffff	(1400)	Rotate Right f	RRF	f, d	f(n)→d(n−1), f(0)→C, C→d(7)	C
001101	d	fffff	(1500)	Rotate Left f	RLF	f, d	f(n)→d(n+1), f(7)→C, C→d(0)	C
001110	d	fffff	(1600)	Swap halves f	SWAPF	f, d	f(0−3)⇄f(4−7)→d	None
001111	d	fffff	(1700)	Increment f, Skip if Zero	INCFSZ	f, d	f+1→d, skip if zero	None

BIT-level file register operations

(8–11)	(5–7)	(0–4)
OP CODE	b (BIT #)	f (FILE #)

Instruction (octal)				*Name*	*Syntax*		*Operation*	*Status*
0100	bbb	fffff	(2000)	Bit Clear f	BCF	f, b	0→f(b)	None
0101	bbb	fffff	(2400)	Bit Set f	BSF	f, b	1→f(b)	None
0110	bbb	fffff	(3000)	Bit Test f, Skip if Clear	BTFSC	f, b	Bit Test f(b); skip if clear	None
0111	bbb	fffff	(3400)	Bit Test f, Skip if Set	BTFSS	f, b	Bit Test f(b); skip if set	None

Literal and control operations

(8–11)	(0–7)
OP CODE	I (LITERAL)

Instruction (octal)			*Name*	*Syntax*		*Operation*	*Status*
1000	00000000	(4000)	Return	RET	-	0→W, RAR→PC	None
1000	kkkkkkkk	(4000)	Return and place Literal in W	RETLW	k	k→W, RAR→PC	None
1001	kkkkkkkk	(4400)	Call subroutine†	CALL	k	PC+1→RAR, k→PC	None
101x	kkkkkkkk	(5X00)‡	Go To address	GOTO	k	k→PC	None
1100	kkkkkkkk	(6000)	Move Literal to W	MOVLW	k	k→W	None
1101	kkkkkkkk	(6400)	Inclusive OR Literal and W	IORLW	k	k∨W→W	Z
1110	kkkkkkkk	(7000)	AND Literal and W	ANDLW	k	k∧W→W	Z
1111	kkkkkkkk	(7400)	Exclusive OR Literal and W	XORLW	k	k⊻W→W	Z

†The 9th bit of the program counter in the PIC1650 is zero for a CALL and a MOVWF F2. Therefore, subroutines must be located in page 0. However, subroutines can be called from page 0 or page 1 since the RAR is 9 bits wide (Page 0: 0–255. Page 1: 256–511).

‡If x = 0, the address is in page 0; if x = 1, the address is in page 1. The PIC assembler takes care of assigning the correct op codes.

DECFSZ is skipped; if the reslt is not zero, then the next instruction is executed. *Increment file, skip if zero* (INCFSZ) operates in a similar fashion.

Instruction Format II: Bit-Level Register Operations

OP CODE	B (BIT #)	f (FILE #)
11 10 9 8	7 6 5	4 3 2 1 0

This format has a 5-bit register select field, a 3-bit bit select field and a 4-bit op code field. There are only four instructions in this category: two instructions that set or clear individual bits in a particular register, and the other two instructions that test a bit for a one or zero and skip accordingly. Again, the register select field (bits 0–4) can directly address one of 32 file registers while the bit select field (bits 5–7) selects one of eight bits in that register to be either set or cleared. The other seven bits in the register remain unchanged. None of the status bits are altered by any of the instructions in this category.

Any instruction in the above two classes may specify F0 in the register select field. In that case, as an example, say the PIC is to execute DECF F0, W and the contents of F4 at the time of execution are 14_8. Then register 14_8 is decremented and its results placed in the W register.

Instruction Format III: Literal and Control Operations

OP CODE	l (LITERAL)
11 10 9 8	7 6 5 4 3 2 1 0

This format has a 4-bit op code and an 8-bit constant field. Instructions using this format fall into two sub-groups: one group treats constants located in program ROM as data, while the other treats them as addresses. As an example of the first, IORLW Inclusive ORs an 8-bit constant with the present contents of the W register and places the result in the W register.

The CALL instruction and the GOTO instruction treat the literal field as an address. Although one GOTO mnemonic appears in Table 2 there are really two GOTO instructions. The program counter is 9 bits wide to address 512 words, while the literal field is only 8 bits wide. Thus, bit 8 of the GOTO instruction specifies the ninth bit of the program counter. The op code field is limited, however, and therefore this same technique cannot be applied to the CALL instruction. Thus, when a CALL instruction is executed, the ninth bit of the PC is forced to a zero. This requires all subroutines to be located in the low 256-word ROM memory space. Similarly, when the PC is also changed by the use of the MOVW F2 instruction, the ninth bit is also forced to a zero. Since the pushdown stack is 9 bits wide, though, subroutines can be called from anywhere in the 512-word ROM space.

The Return and Place Literal in W instruction (RETLW) is a little unusual in that it is two instructions in one. The op code specifies that the top element of the pushdown stack (indicated as RAR for Return Address Register in Table 2) replace the program counter. Simultaneously, the constant contained in the literal field is loaded into the W register. This instruction provides a very convenient facility for table look-up.

V. Implementation

Timing

The basic instruction cycle timing for the PIC is generated from an on-chip ring oscillator. The only external components required to support oscillation are a resistor and a capacitor. The oscillator runs at four times the internal clock frequency; thus, to support a 4-μs instruction cycle, the oscillator must operate at 1 MHz.

An internal two-phase, non-overlapping clocking scheme is central to the microcomputer's internal operation. This is shown in Fig. 2. To keep the control logic simple, a pipelined instruction fetch/instruction execute system was used. Thus, while the PC is accessing the current instruction in ROM memory, the ALU and register array data sections are executing the instruction accessed in the previous cycle. This requires the use of a separate incrementer for the PC as there is no time for the ALU to perform the incrementation. The program counter increments on the

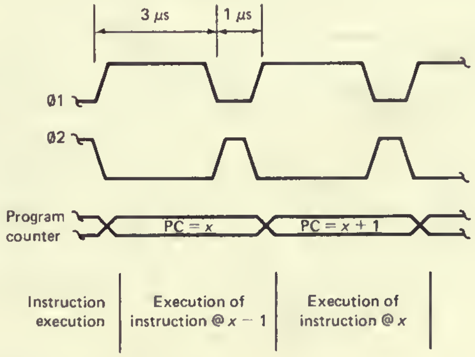


Fig. 2. Instruction cycle timing.

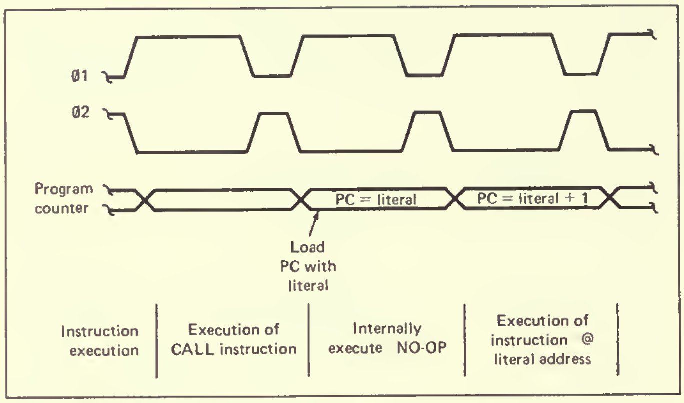


Fig. 3. Modified cycle timing for CALL, GOTO, etc.

rising edge of every ØI clock. At the same time, a master-slave flip-flop located at the output of the ROM latches the instruction fetched in the previous cycle. This prevents the new instruction fetch from potentially corrupting the previous fetch. This pipelining scheme keeps the instruction throughput high.

For those instructions that modify the contents of the program counter, this scheme does not work. Opportunity must be given for the ROM to access the instruction at the new address. Thus the PIC must wait an additonal cycle before accessing the next instruction after CALL or similar instructions (Fig. 3).

Input/Output Registers

Thirty-two pins of the PIC1650 (housed in a 40-pin dual in-line package) constitute the input/output pins. They are segregated into four groups of eight pins each. Each group of eight represents a register that occupies an address in the address space of the register file array. Pins RA<0:7> are the I/O pins that constitute F5 (Fig. 1). Similarly, RB<0:7>:= F6<7:0>,RC<0:7>:= F7<7:0>, and RD<0:7>:= F8<7:0>. A circuit diagram of the I/O register interfacing to a TTL gate is shown in Fig. 4.

Each I/O bit contains a latch which will be written into if its I/O file is specified as the destination register in an instruction. If we consider the I/O bit as part of an output file, then the logic value attained by the pin will be the logic value in the latch. If a "1" is stored in the latch, transistor Q_1 will still be on, hit transistor Q_2 will attain an impedance of approximately 200 Ω and drive the pin to a low level.

An auxiliary power supply, V_{XX}, provides the voltage required to turn on transistor Q_2. The higher this supply, the lower the impedance Q_2 attains when it is on. Typically, increasing the V_{XX} supply from 5 V to 10 V will roughly halve the impedance of Q_2 from 200 Ω to 100 Ω. In driving large-segment LED displays, for example, a typical system configuration would call for V_{CC} (primary chip supply) to be 5 V and V_{XX} to be 10 V. This provides the large current-sinking capability necessary to drive the displays without the need for any interfacing bipolar transistors.

Now consider the use of the I/O bit as part of an input file. When an I/O file is used as a source register, an internal READ signal gates the data on the I/O pin into the internal data bus. In this configuration, Q_2 should be kept off by presetting the register to I (allowing Q_1 to be conveniently used as a pull-up transistor). If Q_2 is on, an impedance conflict will occur if an external device is attempting to drive the pin to a logic "1" level. For purposes of logic definition, then, it can be said that the I/O bit and the external device form a logical AND when the I/O register is used as an input file.

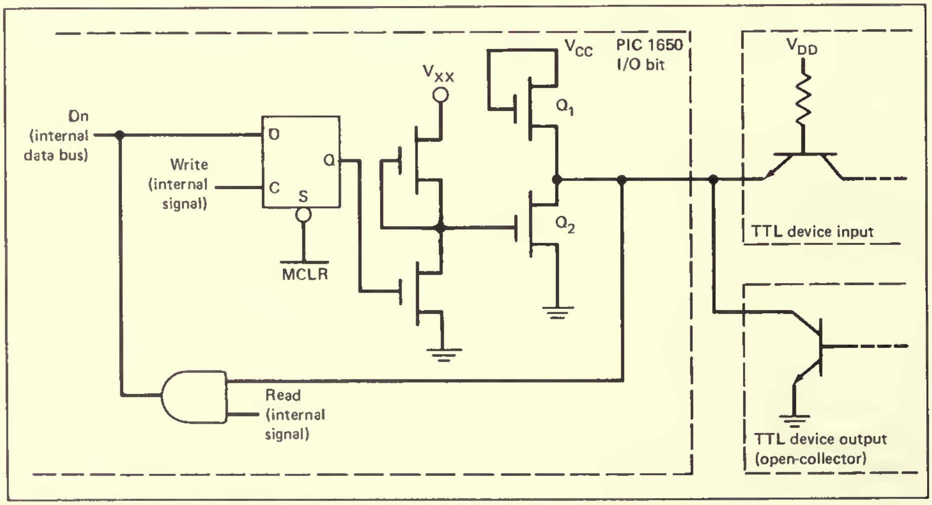


Fig. 4. I/O register circuit diagram.

VI. Program Examples

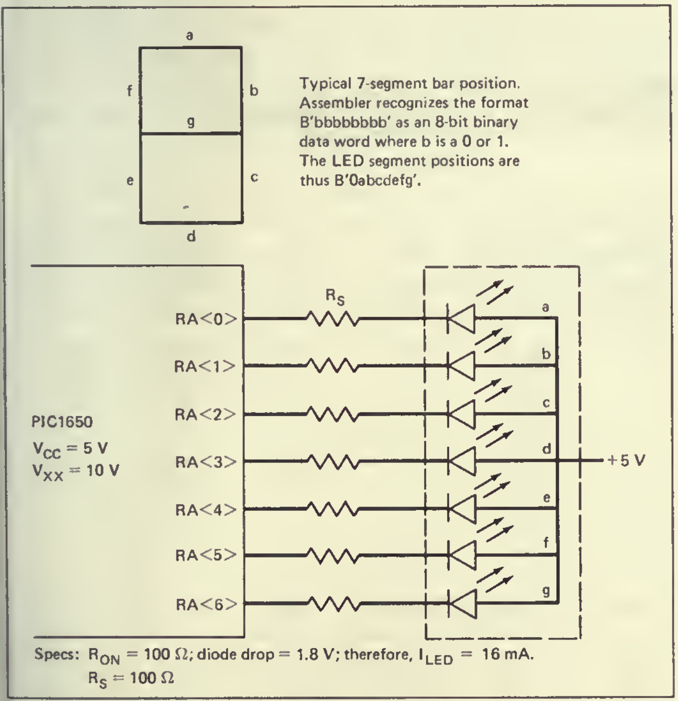
Use of Indirect Addressing

This example illustrates the use of the File Select Register (F4) and the indirect addressing mode using F0. This program clears files F5 to F31.

Label	*Op code*	*Operand*	*Comment*
	MOVLW	5	Move literal 5 to W REG.
	MOVWF	4	Move W to F4. (F4 = 5).
Loop	CLRF	0	Clear the contents of the file pointed to by F4.
	INCFSZ	4, F	Increment F4. The PC will skip after F31 is cleared.
	GOTO	Loop	Repeat the steps beginning at Loop to clear the next file.
	END		Files F5 and F31 are cleared.

BCD Number Display

This example converts a BCD number held in the four least significant bits of F20 (the 4 MSB's are assumed zero) to a 7-segment code. The 7-segment code is output via I/O port F5, which is directly tied through current-limiting resistors to a 7-segment LED display. This program illustrates the use of the computed GOTO instruction. Figure 5 shows the external component connections.



Typical 7-segment bar position. Assembler recognizes the format B'bbbbbbbb' as an 8-bit binary data word where b is a 0 or 1. The LED segment positions are thus B'0abcdefg'.

Specs: $R_{ON} = 100\ \Omega$; diode drop = 1.8 V; therefore, $I_{LED} = 16$ mA. $R_S = 100\ \Omega$

Fig. 5. LED display connection diagram.

Label	*Op code*	*Operand*	*Comment*
	MOVLW	TBLSTR	Starting address of table.
	ADDWF	20, W	Add BCD number as offset to Table start.
	CALL	CONVRT	Call the conversion subroutine.
	MOVWF	5	Output the 7-segment code to I/O F5. The 7-segment will now show the BCD number and this output will remain stable until F5 is set to a new value.
	END		
CONVRT:	MOVWF	2	Move the computed address into the PC. Because the ninth bit of the PC is set to zero by a MOVWF 2, the TBLSTR routine must be located in the low 256-word ROM memory area.
TBLSTR:	RETLW	B'00000001	Complement of 0 in 7-segment code.
	RETLW	B'01001111	Complement of 1 in 7-segment code.
	RETLW	B'00010010	Complement of 2 in 7-segment code.
	RETLW	B'00000110	Complement of 3 in 7-segment code.
	RETLW	B'01001100	Complement of 4 in 7-segment code.
	RETLW	B'00100100	Complement of 5 in 7-segment code.
	RETLW	B'01100000	Complement of 6 in 7-segment code.
	RETLW	B'00001111	Complement of 7 in 7-segment code.
	RETLW	B'00000000	Complement of 8 in 7-segment code.
	RETLW	B'00001100	Complement of 9 in 7-segment code.

The RETLW instruction loads the W register with the specified literal value and returns to the instruction following the CALL instruction (MOVWF 5). The complement of the 7-segment code is used because the LED display unit is common-anode; a segment is activated when the output is set low.

References

PIC [1979*a*]; PIC [1979*b*].